**Abstract**

Text mining refers to the process of transforming unstructured text data into structured clusters of information. This project explores a specific aspect of text mining known as authorship attribution, which involves analyzing linguistic and stylistic features of text to predict its author. This project uses transcription data from South African State of the Nation Address (SONA) speeches, delivered by presidents between 1994 to 2023. The primary goal was to develop a classification model that takes a sentence from a SONA speech as input and correctly predicts the president who said it. Various models were trained, such as Classification Trees, and Random Forests, in addition to XGBoost- , Naïve Bayesian- and Feed Forward Neural Network models. I find that the neural net model, with TF-IDF features outperformed all other models with a test F1-score of 0.632 and a test accuracy of 0.639.

**Introduction**

Text mining is a branch of Artificial Intelligence (AI) that aims to transform unstructured text data into structured formats, and aims to uncover textual patterns within unstructured data [@ibm2023].

Modern text mining employs machine learning methods and Natural Language Processing (NLP) e.g., part-of-speech tagging [@greenbook2017]. Advancements in text mining is largely attributed to the exponential growth in data, with approximately 80% of global data residing in unstructured formats. This vast amount of data necessitated the use of text mining in data analytics.

Text mining is particularly useful in large organizations where decision-making is central and time is limiting [@ibm2023]. The many applications of text mining has led to the development of several models, including supervised, and unsupervised methods [@dogra2022]. However, the choice of the most effective model is a complex task.

The aim of this project is to identify the most effective classification model to predict which present said a specific sentence. The models are trained on transcription data that includes the State of the Nation Address (SONA) speeches delivered by presidents between 1994 to 2023 [@govza2023]. SONA serves as an annual opening to South African Parliament, where the President reports on the socio-economic state of the nation to a joint sitting of Parliament. The main objective is to train a model that has relatively good test- F1-score and accuracy.

**Literature Review**

Several studies have used machine learning methods for author classification [@boukhaled2017]. These techniques include Logistic Regression (LR), Random Forests (RF), and Neural Networks (NN). The choice of model depends on the nature of the data and the specific task requirements.

Feature selection plays an important role in author classification, where features are broadly categorized into lexical features (e.g., word usage) and syntactic features (e.g., part-of-speech tags). Some studies have also used semantic features (e.g., topics and sentiments).

An article by Shukri, (2021) [@shukri2021] suggests a method for author classification, by training models on 8109 articles from 428 authors for the period 2016 to 2021. Their NN model achieved the highest accuracy of 81.1%.

Bauersfeld et al., (2023) [@bauersfeld2023] proposed a transformer-based, neural-network architecture that uses text content and author names in the bibliography to determine the author of an anonymous manuscript. The authors used all research papers publicly available on arXiv and achieved a 73% accuracy rate.

A similar paper by Khalid (2021) [@khalid2021] performed author prediction on 210 000 anonymous, news headlines. The study used Bag-of-Words (BoW) and Latent Semantic Analysis (LSA) features as input to train an LR and RF. The study found that the LR model outperformed all other models with an accuracy of 94.9%.

These papers highlight the variety of author-classification techniques available. However, we also note the challenges in applying these techniques, such as the need for large datasets, as well as the ability of these models to discriminate between content-related features and author-specific features.

**Data**

**Data Source and Description**

The SONA dataset is publicly available on the South African government website [@govza2023]. The data contains the speeches delivered by South African presidents at the annual State of the Nation Address (SONA) from 1994 to 2023.

The following data is available:

* Mandela (1994-1999) - 7 speeches
* Mbeki (2000-2008) - 10 speeches
* Zuma (2009-2017) - 10 speeches
* Ramaphosa (2018-2023) - 7 speeches
* Two outliers exist:
  + deKlerk (1994) - 1 speech
  + Motlanthe (2009) - 1 speech

These records cumulatively formed the SONA dataset with 36 records and 5 variables, namely: *filename, speech, year, president* and *date of speech delivered*.

**Data Pre-processing**

All data was read into R *version 4.3.1* , using R-Studio *version 2023.9.1.494*. The year of each speech was extracted from the first four characters in the *filename* column. The president names were extracted from the *filename* column using regular expressions, where alphabetical text ending in a ".txt" extension was matched as the presidents' name. Subsequently, all other unnecessary text such as "http"-, fullstop-, ampersand-, greater-than-, and less-than characters were removed, in addition to trailing white spaces and new-line characters. Dates were then re-formatted into a *dd-mm-yyyy* format. Finally, the pre-processed data was saved as an RDS object for downstream analysis.

**Methods**

**Data processing**

The pre-processed data was read into R and converted to a tibble. The data was then assessed by looking at the head and tail of the tibble, in addition to looking at the data types of each column. Dates which appeared before a president's speech were removed using regular expressions. Trailing white spaces before and after a speech were also removed. I also noticed that former president- Motlanthe and deKlerk only had one record and so these observations were removed from the dataset.

As a result, the processed data had 34 rows and 5 columns.

**Tokenization**

The processed SONA dataset was then tokenized into sentences using unnest\_tokens(), since the goal is to make predictions on sentence inputs. All text was then converted to lowercase to remove word redundancy. I also removed all punctuation so that root-words are treated alike*.* I then included a sentence ID column to track sentence membership.

The sentence tokens were then tokenized into word and bigram tokens respectively, where all stop words were removed. I also ensured that "blank" tokens were removed. For the tokenization by bigram implementation, bigrams were first split into individual words, where each word was assessed for stop words. If a stop word was detected, the entire bigram was removed, while the remaining bigrams were unified.

**Exploratory Data Analysis**

I looked at the 20 most frequently used- words and bigrams: *(1)* for all presidents, and *(2)* for each president. I also looked at the average number of sentences per speech and the average number of words per sentence for each president.

**Features**

**Bag-of-Words Model**

The Bag-of-Words (BoW) model computes the frequency of occurrence of an unordered collection of words within a document and uses these frequencies as features to train the classifier.

A word bag was generated by taking the word tokens and grouping the unique combinations of "sentence ID, president and word" and computing each grouping's frequency, after which the top 200 words for each grouping was selected to create the final word bag. The choice of the top 200 words was chosen due to its superior model performance relative to the top 100- and 500 word models (tested informally). The word bag consisted of 363 rows (words) and 3 columns, namely, *sentence ID, president, and word.*

The BoW table was then constructed by identifying all the words in a sentence that overlap with the word bag. The frequency of each word within a sentence was then calculated. Finally, the BoW table was reformatted to a tidy format, where columns represent the features (149 words), the rows represent the observations (sentence ID) and the cell values are the frequency of a word within a sentence. All words not found in a sentence obtained a value of 0.

**Term Frequency-Inverse Document Frequency Model**

Term Frequency – Inverse Document Frequency (TF-IDF) refers to the metric that describes how important a word is in a document relative to other documents in the corpus. TF-IDF is calculated by multiplying the Term Frequency (TF) by the Inverse Document Frequency (IDF), where TF is the frequency of a word within a document divided by the total number of words in that document, whereas IDF is the logarithmically scaled inverse fraction of the documents that contain the word. Bind\_tf\_idf() was used to calculate the TF-IDF for each word in each sentence. It is important to note that the "document" here refers to the sentence ID. The BoW table previously discussed was then manipulated to include TF-IDF values instead of word frequencies.

**Class Imbalance and Up-Sampling**

I checked for class imbalance by comparing the frequency of records in each of the target variable classes, namely: *Mandela, Mbeki, Zuma* and *Ramaphosa*. I found that the classes were imbalanced, where Mbeki had the largest proportion of sentences (92), followed by Ramaphosa (51), Zuma (41) and Mandela (21). As a result, I used upSample() in the Caret package to oversample the minority classes so that the number of observations in each class matched the majority class.

**Split Balanced Data into Training, Validation and Test sets**

I partitioned the data into 70% training and 30% test sets using createDataPartition(), which performs stratified sampling. The target variable, *president* was then converted to factors, where classes: *Mandela, Mbeki, Ramaphosa* and *Zuma* were categorized as levels 1 to 4, respectively. For the validation set, 5-fold cross-validation was applied during training to determine the optimal model parameters by means of a grid-search (discussed below).

**Workflow**

Each classification model implemented both the BoW and TF-IDF features discussed above. For each model, a grid search was performed to find the optimal hyperparameters, generating sub-models as a result. For each sub-model, 5-fold cross-validation was performed on the training set. The best sub-model was determined by the model performance on both the training and validation sets. The hyperparameters of the best sub-model was then used to re-train the model on the full training set, after which predictions were made on the test set. The final models were compared based on their test set performance. @fig-1 below shows the general workflow.

In **Figure 1** above, the ribbons are the steps in the process and the rectangles are the choices made at each step.

**Model Construction**

**Classification Tree**

Classification trees recursively partition the input space and assigns a class label to each partitioned region based on the majority observation classes in that region. A grid search was performed using the rpart() function with the following parameters:

* The complexity parameter, a stopping criterion where tree splitting terminates once the reduction in relative error is less than a specified *cp-threshold :*

**cp** = {0.001, 0.112, 0.223, 0.334, 0.445, 0.556, 0.667, 0.778, 0.889, 1.000}

* The minimum number of observations at any terminal node:

**minbucket** = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

* The minimum number of observations that must exist in a node for a split to be attempted:

**minsplit** = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

**Random Forest**

Random Forests use bagging to train multiple decision trees in parallel, reaching a majority vote of class classification. This type of ensemble learning helps to overcome overfitting.

The randomForest() function was used, taking in the following parameters:

* The number of trees to grow:

**ntree** = {100, 500, 1000}

* Feature importance:

**Importance** = TRUE

* **na.action** = na.exclude

**Extreme Gradient Boosting**

Extreme Gradient Boosting (XGBoost) models aggregate the output of a sequential ensemble of tree models, where each subsequent model improves on the previous model. The xgboost() function was used, with the following *important* parameters:

* The maximum depth of the tree, where increasing this value results in a more complex model:

**max depth** = {1, 2, 3, 4, 5, 6}

* The step size shrinkage, which is used to prevent overfitting. This value of *eta* shrinks the feature weights to make the boosting process more conservative:

**eta** = {0.2, 0.4, 0.6, 1}

* The minimum loss reduction, where the larger the value of *gamma* is, the more conservative the algorithm is:

**gamma** = {0.5, 1}

* The L2 regularization term on the weights:

**lambda** = 1

* Objective function for multi-class classification:

**objective** = "multi:softmax"

It is important to note, that the labels of the input data were converted to factors then to integers and finally offset by -1 because xgboost() takes in numeric data where classes of the target variable should be indexed from 0. The features need to be converted to a DMatrix object using the xgb.DMatrix() function. Consequently, I offset the predictions by +1 when computing the confusion matrix, since R indexes from 1. I then had to convert the numeric predictions back to factors, where:

* 0+1: Mandela
* 1+1: Mbeki
* 2+1: Ramaphosa
* 3+1: Zuma

**Naïve Bayes**

Naïve Bayes is a generative model that seeks to model the distribution of inputs of a given class. The naiveBayes() function was used, while applying Laplace smoothing to handle zero probabilities in categorical data. Laplace values of **0, 0.1 and 1** were investigated.

**Feed Forward Neural Network**

A feed-forward neural network is a type of artificial neural network where the connections between nodes do not form a cycle. Information in this network moves only in one direction, forward, from the input layer through the hidden layers, to the output layer.

A feed-forward neural network was built using Keras and TensorFlow within a Conda environment managed by Anaconda, all executed in RStudio. The target variable (*president*) was extracted from the training data, and converted to factors, after which the factors were converted to discrete integers from 0 to 3, where:

* 0: Mandela
* 1: Mbeki
* 2: Ramaphosa
* 3: Zuma.

Subsequently, the target variable *president* was one hot-coded to a binary representation. The features were then extracted and converted to a matrix format. The following hyperparameters were used:

* **Input neurons =** {50, 100, 200}
* **Learning rate =** {0.001, 0.01, 0.1}
* **Drop-out rate =** {0.01, 0.1}

The network was built as follows:

* **Layer 1:** a dense, fully connect layer with *x* input neurons specified above, and which uses either a Tanh or ReLU activation function. This layer expects 149 predictor variables (words).
* **Layer 2:** a dropout layer to prevent overfitting by randomly selecting a fraction (drop-out rate above) of the input units at each update and setting them to 0 during training time.
* **Layer 3:** a dense, fully connected layer with 4 output neurons (representing the 4 president classes). A softmax activation function was used to generate a vector of probabilities of class membership for each observation.

The categorical cross-entropy loss function was minimized which is invariant to shifting of the predicted probabilities. The n-Adam optimizer was used to update the model parameters (weights and biases). This choice of optimizer was based on its superior performance compared to other optimizers, such as SGD, RMSprop, and Adam (informally tested). Finally, the data was trained for 30 epochs, with a batch size of 5, while shuffling the training data at each epoch.

**Model Performance**

For all models, performance on the training, validation and test sets were determined by the metrics discussed below. For 5-fold cross-validation, the best sub-model was determined which had the best mean cross-validation (CV) and training macro-average F1-score. The final model was chosen based on the model with the best test set macro-average F1-score. Other metrics were also considered (see below).

At each fold, the metric is computed for every class, and the macro-average metric is obtained by averaging these class-specific metrics (**Equation 1**). After training is complete, the final metric is computed as the average of the macro-average metrics over all folds (**Equation 2**).

**Accuracy**

The proportion of correct classifications among the total number of classifications.

*…equation 3,* where TP, TN, FP and FN are the number of true positives, true negatives, false positives, and false negatives, respectively.

**Recall**

Also known as the sensitivity, this is the proportion of actual positives that are correctly classified.

**Precision**

Also known as the positive predictive value, this is the proportion of positive predictions that are correctly classified.

**F1-score**

I chose the F1-score as my main criterion for model performance because the F1-score provides information on the model's ability to capture positive cases (recall) and be accurate with the cases it does capture (precision).

**Results**

**Exploratory Data Analysis**

**Top 20 most frequently used words and bigrams**

In @fig-2 below we see the top 20 most frequently used- **(a)** words and **(b)** bigrams, among all presidents. The bigrams provide more context relative to the unigrams, and we see that most president's agenda is about the economy with bigrams such as "economic growth" and "job creation". @fig-3 below shows the top 20 most frequently used words- and @fig-4 below shows the top 20 most frequently used bigrams- for each president. In these figures we see that the agenda is specific to a president or period of presidency, where Mandela has bigrams such as "people-centered society" alluding to the end of Apartheid, Mbeki focuses on social justice with bigrams such as "social security" and "social partners", Zuma mentions the "world cup" which alludes to the 2010 soccer world cup hosted in South Africa and Ramaphosa's focus is on crime prevention with bigrams such as "gender-based violence" and "law enforcement".

**Average speech and Sentence Length**

**@fig-5** below shows the **(a)** average speech length per president and **(b)** the average sentence length per president. In **@fig-5 (a)** we see that Ramaphosa has a relatively long average speech length (327 sentences per speech), followed by Zuma (266), Mbeki (242) and Mandela (238). In **@fig-5 (b)** we see that Mbeki has the largest average sentence length (30 words per sentence), followed by Mandela (25) , Ramaphosa (22) and Zuma (19).

**Bag-of-Word Models**

**Classification Trees**

In **@tbl-1** below, we see the results of the best performing classification trees for the BoW features.

We see that model 1 and 2 both have the best mean CV F1-score of 0.546 and a training F1-score of 0.749. These models also have the best mean CV accuracy of 0.565 and a training accuracy of 0.751. These models perform better than the other models with regards to the other metrics as well. Model 1 parameters were chosen to re-train the model on the full training set. The results of training can be seen in the BoW tree diagram in @fig-6 .

**Random Forests**

In @tbl-2 we see that all models performed equally well. All metrics have values above 0.6, suggesting good model fit. The mean CV-F1 score = 0.648 and the training F1-score = 0.641. The training accuracy = 0.639 and mean CV accuracy = 0.65. I chose RF sub-model 2 with *ntrees* = 500 to re-train the full training set, which strikes a balance between too many and too few trees.

**Extreme Gradient Boosting**

@tbl-3 below suggests that sub-model 47 had the best model performance, and so the model was re-trained on the full training dataset using max depth = 6, eta = 1 and gamma = 0.5. We see that the training F1-score (0.621) and training accuracy (0.62) is relatively good, however the mean CV F1-score (0.426) and mean CV accuracy (0.438) is relatively low . We see a similar trend for all other metrics, where the training performance is good, but the validation performance is poor, suggesting that the model overfits

**Naïve Bayes**

As seen in **@tbl-4** below, all "top" performing models displayed poor performance. These models all displayed the same metric values, with a training F1-score of 0.106 and a mean CV F1-score of 0.1. The training accuracy of 0.253 and mean CV accuracy of 0.25 also suggests poor model performance. I used the simplest model, model 1 with no Laplace smoothing to re-train the model on the full training dataset.

**Feed Forward Neural Network**

In @tbl-5 below, we see the results of the feed forward neural network, where model 55 had the best training F1-score of 0.857 (as well as model 17) and the best mean CV F1-score of 0.593. We also see that model 55 had one of the best training accuracies (0.852) and the best mean CV accuracy (0.612). All other metrics for model 55 are also among the top performing models.

**@fig-7** in **Addendum A** shows the change in the Log Loss, training- recall, precision and accuracy over all 30 epochs.

**Term Frequency-Inverse Document Frequency Models**

**Classification Trees**

In @tbl-6 below, we see that sub-model 1 and 2 had the best mean CV F1-score of 0.583, training F1-score of 0.742, mean CV accuracy of 0.596, and training accuracy of 0.746. In parallel, we see that that all other metrics for sub-models 1 and 2 are among the best performing models. Model 1 parameters were chosen to re-train the model on the full training set. We see interesting clusters in the TF-IDF tree diagram in **@fig-8** .

**Random Forests**

In **@tbl-7** below we see that the grid-search performed equally well for all *ntree* values. All metrics are relatively good, with values above 0.6, suggesting good model fit. We see that the mean CV-F1 score (0.653) is similar to the training F1-score (0.647). We also see that the mean CV accuracy (0.654) is similar to the training accuracy (0.647). I chose RF sub-model 2 with *ntrees* = 500 to re-train the full training set, which strikes a balance between too many and too few trees.

**Extreme Gradient Boosting**

@tbl-8 below suggests that sub-model 39 has the best model performance, and so this model was re-trained on the full training dataset. We see that the training F1-score (0.637) and training accuracy (0.637) is relatively good, however the mean CV F1-score (0.465) and mean CV accuracy (0.481) is relatively low . We see a similar trend for all other metrics, where the training performance is good, but the validation performance is poor, suggestive of model overfitting.

**Naïve Bayes**

As seen in @tbl-9 below, all "top" performing models displayed poor performance. These models all displayed the same metric values, with a training F1-score of 0.1 equivalent to the mean CV F1-score of 0.1. The training accuracy and mean CV accuracy are both 0.25 suggesting poor model performance. I used the simplest model, model 1 with no Laplace smoothing to re-train the model on the full training dataset.

**Feed Forward Neural Network**

In @tbl-10 below, we see that model 42 had the best training F1-score of 0.849 and one of the best mean CV F1-scores of 0.637. We also see that model 42 had among the best training accuracies (0.851) and the best validation accuracy (0.650). All other metrics for model 42 are also among the top performing models.

@fig-9 in **Addendum A** shows the change in the Log Loss, training- recall, precision and accuracy over all 30 epochs.

**Test Set Model Performance**

In @tbl-11 below we see the model performance on the test set. The neural net model 42, using the TF-IDF fetaures outperformed all other models. @fig-10 in **Addendum A** shows the confusion matrix of TF-IDF NN model 42.

**Discussion**

**Training and Validation Model Performance**

For the BoW @tbl-1 and TF-IDF @tbl-6 classification trees we see that the training performance is generally better than the validation performance, suggestive of model overfitting, where the model does not generalize well to the validation data. We see a similar trend for the XGBoost models ( BoW @tbl-3 and TF-IDF @tbl-8 ).

We see two interesting clusters in the tree diagrams in @fig-6 (BoW) and @fig-8 (TF-IDF) of **Addendum A**, which suggests that the unigram "health" is an important feature in the sentences of Ramaphosa (2018-present), which is in accordance with the Covid-19 pandemic. In addition, "peace" is an important feature in the sentences of Mandela (1994-1999), which is in accordance with the end of the Apartheid era (1994).

For the BoW (@tbl-2) and TF-IDF (@tbl-7) Random Forest models, we see a trend where the validation metrics are similar to the training metrics, where all metrics have a value above 0.6 suggesting good model fit. These models generalize well to the validation data.

The Naive Bayesian models (BoW, @tbl-4 and TF-IDF, @tbl-9 ) have poor model fit on the training data, and does not generalize well to the validation data.

For the BoW **(** @tbl-5 and @fig-7 in **Addendum A )** and TF-IDF (@tbl-10 and @fig-9 in **Addendum A)** Neural Network models, we see the log loss decreases to 0, as the number of epochs increase, whereas the training- recall, precision and accuracy values increase towards a maximum value. This suggests that the loss has been minimized and the performance, maximized.

**Test Set Model Performance**

The overall best model (Neural Net model 42, using the TF-IDF features) (@tbl-11), outperformed all other models with a test F1-score of 0.632. This model also had the best test set accuracy (0.639). However, the model's test precision score of 0.653 felt short of the neural net model 55, using BoW features (0.709). In addition, NN model 42 had a test recall of 0.611, which felt short of the random forest model 2 using TF-IDF features, with a recall of 0.620.

The neural net models and the random forest models for both BoW and TF-IDF generalized well to the unseen, test data, with performance metrics above 0.6 in most cases. The classification trees and XGBoost models had sub-optimal performance, which suggests that these models do not generalize as well to the unseen, test data. In addition, the Naïve Bayesian models have very poor model fit and do not generalize well to the unseen, test data. These sub-optimal and poorly fit models may be over-fitting to the training data.

In general, we also see a trend where in most cases the TF-IDF models outperform the BoW models, which may indicate that word importance is a better feature choice than word frequency.

**Conclusion**

In this comparative analysis of different classifiers and features, the neural net model with TF-IDF features demonstrated the highest performance in terms of test F1-score and accuracy. In addition, the random forest models also showed good generalization to unseen data. Notably, models trained on TF-IDF features performed better than those using BoW features. In future, one should consider additional features such as the average number of words used in a sentence by each president, along with sentiment and topical features, to potentially improve the model’s performance.